Chapter 7

CACHE REPLACEMENT ALGORITHMS

The efficiency of proxy caches is influenced to a significant extent by document placement/replacement algorithms. *Cache placement algorithms* try to place documents within a cache whereas *cache replacement algorithms* replace documents from a cache. Such algorithms are beneficial only if they can ameliorate the hit ratio (see Glossary for a definition of hit ratio). In contrast to cache replacement, the subject of cache placement has not been well researched.

Cache replacement algorithms often aim to minimize various parameters such as the hit ratio, the byte hit ratio, the cost of access and the latency (refer Glossary for definitions).

The most widely used cache replacement algorithms include Least Recently Used (LRU), Least Frequently Used (LFU), LRU-Min [5], LRU-Threshold [5], Pitkow/Recker [447], SIZE [552], Lowest Latency First [564], Hyper-G [552], Greedy-Dual-Size (GDS) [102], Hybrid [564], Lowest Relative Value (LRV) [467], LNC-R-W3 [492], Bolot/Hoscka [80], Size-adjusted LRU (SLRU) [11], Least Unified-Value (LUV) [38], and Hierarchical Greedy Dual (HGD) [312]. Several of these attempt to maximize the hit ratio.

There are several ways to categorize cache replacement algorithms (see [448] for a representative categorization and for a survey of Web cache replacement strategies). We may follow the approach suggested by Aggarwal et al. (see [11]):

- Traditional algorithms
- Key based algorithms
- Cost based algorithms

Table 7.1. A list of some popular cache replacement algorithms

Least Recently Used (LRU)
Least Frequently Used (LFU)
LRU-Min
LRU-Threshold
Pitkow/Recker
SIZE
Lowest Latency First
Hyper-G
Greedy Dual Size (GDS)
Hybrid
Lowest Relative Value (LRV)
LNC-R-W3
Bolot/Hoscka
Size-adjusted LRU (SLRU)
Least Unified-Value (LUV)
Hierarchical Greedy Dual (HGD)

1. Traditional algorithms

Traditional cache replacement algorithms include LRU, LFU, Pitkow/Recker and some of their variants.

Least Recently Used (LRU) expels the object from the cache that was asked for the least number of times, of late. Least Frequently Used (LFU) expels the object that was retrieved least frequently from the cache. Pitkow/Recker [447] expels objects in LRU order unless all objects are referenced in a single day, in which case the biggest object is replaced.

It is quite possible that some cache replacement algorithms may not perform well when certain parameters such as hit ratios, size etc. are used for evaluating them. Williams et al. [552] mention that proxy servers can possibly diminish the following three quantities: the number of requests sent to favorite servers, the volume of network traffic produced as a result of requests for documents, and the access latency. They look at the first two by employing two metrics: cache hit ratio and byte hit ratio (see Glossary for definitions of these terms). Williams et al. present a taxonomy of cache retrieval policies. By means of trace-driven simulations they measure the maximum feasible hit ratio and byte hit ratio. They conclude that the norms employed by LRU and Pitkow/Recker have the worst performance in their simulation. They suggest that it would be much better to replace documents based on the size as this maximizes the hit ratio in each of their workloads.

2. Key based algorithms

Key based replacement algorithms expel objects from caches on the basis of a primary key. Ties may be broken by the use of secondary or tertiary keys or in other ways. The algorithm SIZE [552] expels the object having the largest size from the cache. LRU-Min [5] favors smaller objects. If there happen to be any objects in a cache with size at least S then LRU-MIN expels (from the cache) the least recently used object satisfying that condition. However, in case there are no objects with size at least S then LRU-Min begins expelling objects of size at least S/2 in LRU order. See Abrams et al. [5] for more details regarding LRU-Min. Abrams et al. also describe LRU-Threshold which is essentially the same as LRU except that objects that are larger than a certain size are never cached.

The Hyper-G cache replacement algorithm described by Williams et al. [552] is an enhancement of LRU. In Hyper-G, ties are broken by checking the recentness of earlier access and also by measuring the size of objects.

Wooster [565] considers optimizing response time instead of hit ratios when studying Web proxy caches. Wooster mentions that several previous studies concerning proxy caches have focused primarily on bettering the cache hit ratio. However, doing so would brush aside the actual time taken to fetch files. Wooster studies cache replacement algorithms that take into account the time taken to fetch a file as one of the factors. Wooster mentions that a cache replacement algorithm that attempts to minimize only the time taken to download a file does not show good performance. An algorithm that overcomes this shortcoming is then developed. It makes use of the following three factors:

- the number of hits of a file
- the size of the file being fetched
- the rate at which a file is fetched.

Wooster compares the performance of the two algorithms mentioned above alongside other popular cache replacement algorithms such as the Least Recently Used (LRU) cache replacement algorithm, the Least Frequently Used (LFU) cache replacement algorithm and a cache replacement algorithm based on the size of a document known as SIZE [552]. Wooster reports that the three factor algorithm mentioned above performs well while diminishing the average latency observed by users.

Wooster and Abrams [564] try to find out whether users do not have to wait for long if proxy caches make use of information about the prevailing network conditions in cache replacement algorithms. They study the two algorithms described in Wooster's thesis [565] (mentioned in the previous paragraph).

They discuss Lowest Latency First. This algorithm keeps the average latency to a minimum by expelling first the document with the lowest download latency. Wooster and Abrams design two cache replacement algorithms that:

- store documents that take the maximum amount of time to fetch
- make use of a combination of multiple requirements such as maintaining in the cache documents from servers that take significant time to connect to, those that need to be fetched from the slowest links, those that have been accessed very frequently, and those that are small.

The first algorithm is known as Lowest Latency First whereas the second one is known as Hybrid. The two algorithms execute by calculating the delay in downloading Web pages or the proxy to server bandwidth by making use of pages recently fetched. Wooster and Abrams check the performance of these two algorithms alongside LRU, LFU and SIZE. The parameters by which the performance is evaluated are user response time, capability to reduce server loads and bandwidth consumption.

3. Cost based algorithms

The algorithms in this class make use of a cost function based on parameters such as the time since the last access was made, the time at which an object was put into the cache, the time of expiration of the object and so on.

Cao and Irani [102] study cost-aware proxy caching algorithms. They introduce the Greedy-Dual-Size (GDS) cache replacement algorithm. This algorithm integrates locality along with cost and size factors. They mention that trace-driven simulations demonstrate that with the right definition for the cost factor, the Greedy-Dual-Size algorithm does far better than many existing Web cache replacement algorithms in several respects. For example, when compared to such algorithms, it attains high hit ratios and enables superior latency reduction. The Greedy-Dual-Size algorithm is stated to have the capability for bettering the functioning of main-memory caching of Web objects (compare with the results in Markatos [379]).

Greedy-Dual-Size tags a cost with every object and expels the object that has the lowest cost or size. It is also possible to tag a utility function with every object and expel the object that is least useful for diminishing the total latency. This approach is used by the Hybrid cache replacement algorithm discussed by Wooster and Abrams [564] (mentioned earlier).

The Lowest Relative Value (LRV) cache replacement algorithm discussed by Rizzo and Vicisano [467] expels the object that has the lowest utility value. In LRV, the utility of a document is calculated adaptively on the basis of data readily available to a proxy server. Rizzo and Vicisano show that LRV performs

better than LRU and can substantially better the performance of a cache that is of modest size.

Scheuermann et al. [492] argue that majority of the cache replacement algorithms do not take into account the large scale of the Web. They discuss the design of delay-conscious cache replacement algorithms that take this fact into consideration by preferentially caching documents that take a long time to get into a cache. They present experimental results concerning a delay-conscious cache replacement algorithm known as LNC-R-W3 (Least Normalized Cost replacement for the Web) that maximizes a performance metric named as *delay-savings ratio* by them. Scheuermann et al. compare the performance of their algorithm with LRU and LRU-Min. LNC-R-W3 uses a rational function of the access frequency, the transfer time cost and the size. In another article [501], Scheuermann and collaborators describe a cache replacement algorithm known as LNC-R-W3-U (LNC-R-W3 with updates) that merges both cache consistency and cache replacement capabilities.

Bolot/Hoscka is a cache replacement algorithm proposed by Bolot and [80]. It uses a weighted rational function of the transfer time cost, the size, and the time of previous access. Size-adjusted LRU (SLRU) developed by Aggarwal et al. [11] arranges objects by the cost to size ratio and picks objects with the most advantageous ratio.

Cohen et al. [132] evaluated server-assisted cache replacement techniques. They represent the utility of caching an object by means of the cost for fetching it, its size, the next request time, and cache prices during the time period between requests. They expel the object of least utility.

Bahn et al. [38] develop a cache replacement algorithm known as Least Unified-Value (LUV). It assesses a Web document on the basis of its cost after normalization by the probability of it being accessed again. Bahn et al. state that this leads to an equitable cache replacement policy. LUV has the ability to adjust to discretionary cost functions of Web documents, as a result of which, it is able to optimize desired quantities such as the hit rate, the byte hit rate, or the delay-savings ratio. LUV makes use of the entire reference history of a document taking into account factors such as the recentness and the frequency of reference, in order to calculate the probability of it being referenced again. Although, it might appear that this could result in significant time and space overheads, surprisingly, LUV may be implemented in an efficient manner with respect to both time as well as space. The time complexity of the algorithm is $O(log_2 n)$, where n is the number of documents in the cache, whereas, the space required to deal with the reference history of a document is only a few bytes. Bahn et al. claim that the LUV algorithm has better performance than prevailing cache replacement algorithms with respect to several performance criteria.

Korupolu and Dahlin [312] discussed a cache replacement scheme known as Hierarchical Greedy Dual (HGD). HGD performs both cache placement and replacement in a collaborative manner in a cache hierarchy.

Before we conclude this chapter, we must note that the performance of cache replacement algorithms depends to a great extent on the nature of Web accesses. Rabinovich and Spatscheck [455] argue that limited progress in developing cache replacement algorithms would not have a substantial impingement on proxy caching.

4. Further reading

Partl and Dingle [431] compared the efficiency of Web cache replacement algorithms and also discussed the advantages of a shared cached space. They compared four popular cache replacement algorithms along with their own cache replacement algorithm known as STCWS. They mention that their algorithm performed well when compared to the four algorithms studied by them.

Bolot et al. [81] discuss the design of efficient caching schemes for the World Wide Web. They argue that many of the algorithms that were developed for Web caching were initially developed for memory caches and thus were not really designed for Web caches. Such algorithms tend to have some major drawbacks. Bolot et al. develop a cache replacement algorithm that minimizes the document retrieval latency as perceived by the user and takes into account document transfer times and sizes of documents.

Mogul et al. [398] investigate the possible advantages of employing *delta-encoding* and data compression for Web cache replacement policies. Many caching methods replace the contents of documents entirely when caches get updated. This is wasteful because only the differences (viz. the delta's) could be moved. Mogul et al. mention that substantial betterment in the response size and latencies could be obtained for a subset of HTTP content types. They mention that data compression could also be employed profitably. They report that merging both delta-encoding and data compression shows much better results than the individual methods. The combination of delta-encoding and data compression is often referred to as delta-compression. Mogul et al. suggest some extensions to the HTTP protocol in order to incorporate both delta-encoding and data compression.

Belloum and Hertzberger [57] study the impact of one-timer documents (documents that are accessed only once or potentially very few times) on Web caches. They study the impact of one-timer documents on a number of cache replacement polices and mention that several such policies often retain one-timer documents for a long time in a cache. As a result, they propose a method for dealing with one-timer documents. They also evaluate their method and study how it is able to deal with one-timer documents. In another article [58], Bel-

loum and Hertzberger study several cache replacement strategies and compare them by means of trace-driven simulations.

Kangasharju et al. [290] discuss the implementation of optimized cache replacement algorithms in a soft caching system (viz. caching by recoding images to lower resolutions). In order to do this, they look at the algorithms suggested for optimized soft caching and test them by employing real proxies.

It may indeed be worthwhile to partition caches in order to gain some benefits. Murta et al. [404] analyze the performance of partitioned caches for the World Wide Web. They mention that although many different solutions exist for Web caching such as:

- client caching
- server caching
- proxy caching
- hierarchical caching and
- co-operative server caching

a problem that appears for all of them is the question of managing cache space effectively. Murta et al. mention that majority of the caching techniques attempt to maximize either the hit ratio or the byte hit ratio. For this reason they focus on these aspects. They suggest that a cache may be divided into partitions. These partitions are permitted to store groups of documents based on their sizes. Murta et al. contrast their strategy with cache replacement strategies that employ algorithms that are either based on the size of documents or make use of the Least Recently Used (LRU) cache replacement algorithm. They mention that their strategy performs well and betters the performance in terms of the hit ratio as well as the byte hit ratio.

Reddy and Fletcher [462] discuss an intelligent Web caching strategy that makes use of the life histories of documents. They mention that maintaining Web pages in either proxy servers or the browser caches of the clients in a hierarchical manner leads to cache consistency difficulties. Consequently, this necessitates the usage of cache maintenance strategies that are effective and also precise in order to ensure good performance. Reddy and Fletcher argue that many of the prevailing techniques such as the Least Recently Used (LRU) cache replacement technique are completely insufficient for handling the heavy loads on networks anticipated in the coming years. They make use of the life histories of documents in order to maximize cache performance. They use a technique named by them as *damped exponential smoothing*. This technique is used for representing in a precise manner the rate at which files are requested and altered. Reddy and Fletcher show that their strategy which employs the life histories of documents does better to a considerable extent than the LRU cache

replacement algorithm. It is stated to achieve this without any loss of efficiency or deterioration of performance. Also refer [460].

In an article related to the above discussion, Reddy and Fletcher [461] compare the performance of a simple adaptive caching agent that makes use of the life histories of documents along with the strategies prevalent at that point of time.

Arlitt et al. [25] assess content management techniques for Web proxy caches. By means of traces of client requests to a Web proxy, they evaluate how various cache replacement policies perform. They introduce an approach for bettering the functioning of a cache while considering several metrics at the same time. They name their technique as *Virtual Caches*.

Gschwind and Hauswirth [221] develop a high performance cache server known as NewsCache for Usenet News (a high traffic news source). They present results of an experimental comparison of several cache replacement strategies.

Dilley et al. [174] discuss improvements to the cache replacement policies of the open source caching software Squid. They describe how they developed two cache replacement policies and enforced them for Squid.

Kelly et al. [300] discuss biased cache replacement policies for Web caches. They mention that based on the needs, it is possible to provide differential quality of service (QoS). For example, different servers may require load reduction to contrasting extents. Kelly et al. present a variation of the Least Frequently Used (LFU) cache replacement algorithm. Their algorithm is receptive to fluctuating levels of server values for cache hits. They mention that on the basis of an assumption concerning server values, their algorithm produces a higher proportion of byte hit ratios for servers that attach more importance or weight for cache hits. They mention that their cache replacement algorithm does more good for servers than either the Least Recently Used (LRU) or the Least Frequently Used (LFU) cache replacement algorithms.

Rochat and Thompson [468] discuss a proxy caching technique based on locating Web objects by taking into account semantic usage. They describe an approach different from that of well-known cache replacement algorithms such as the Least Recently Used (LRU) cache replacement algorithm or the Least Frequently Used (LFU) cache replacement algorithm. They take into consideration characteristics such as object usage, cross-references between documents and geographical location. For their technique, they make use of the following:

- a normalized indexing tree
- an algorithm for getting a better picture of the Web usage patterns
- a data structure for depicting cross-references amongst documents.

Arlitt et al. [26] evaluate the performance of Web proxy cache replacement policies. They study the usefulness of various Web proxy workload characteristics in assisting good cache replacement decisions. They assess workload characteristics such as object size, recency of citation, and frequency of reference. They also discovered long term unwanted secondary effects due to the replacement policies and assessed them. Arlitt et al. conclude from their experiments that higher cache hit rates are obtained by employing size-based replacement policies. Such policies keep numerous small objects in the cache, thereby increasing the likelihood of an object being found in the cache when it is asked for. In order to attain higher byte hit rates, Arlitt et al. suggest that a small number of slightly bigger files should be held in the cache.

Cheng and Kambayashi [123] present a cache replacement algorithm known as LRU-SP. It is an extension to the LRU cache replacement algorithm and takes into account popularity characteristics in addition to size. LRU-SP is built on the Size-adjusted LRU (SLRU) algorithm and another cache replacement algorithm known as Segmented LRU. Cheng and Kambayashi compare the performance of LRU-SP alongside Size-adjusted LRU, Segmented LRU, and the LRV cache replacement algorithm due to Rizzo and Vicisano. They extend their own ideas and discuss cache content management techniques in which contents of caches are distributed among sub-caches. They evaluate their scheme by means of LRU-SP and show that LRU-SP does better than the other three cache replacement algorithms mentioned above.

Jin and Bestavros [273] discuss popularity-aware Greedy-Dual-Size (GDS) algorithms for Web access. They mention that the relative frequency with which objects are sought by the employment of a proxy is rarely taken into account while designing cache replacement algorithms. They state that many cache replacement algorithms often rely on secondary properties that are simpler to obtain, in order to guess popularity characteristics. In their article, they describe:

- an online algorithm for efficiently obtaining popularity information about the Web objects sought by means of a proxy

- a new cache replacement strategy that is an improvement over the Greedy-Dual-Size cache replacement algorithm.

They show that their algorithm performs better than other popularly used algorithms.

Jin and Bestavros [274] describe a Web cache replacement algorithm known as GreedyDual*. This algorithm is a variant of the Greedy-Dual-Size (GDS) cache replacement algorithm. GreedyDual* makes use of metrics that were proposed previously by them. These metrics are used to change the relative value of long-duration popularity when compared to short-duration temporal correlativity of references. They mention that both long-duration popularity

and short-duration temporal correlativity of references for Web cache replacement policies have not been well studied. They had shown earlier that Web reference patterns differ considerably in the prevalence of these two sources of temporal locality. Their caching technique is designed to adapt dynamically to the preponderance of these two sources of temporal locality.

Hamilton et al. [234] study the problem of developing cache replacement policies that are either optimal or near-optimal. It is a fact that the objects that happen to be accessed are often accessed with varying frequencies. In addition, they usually have different retrieval costs when they are not found not in the cache, as they may have to be accessed from servers at varying distances and with different retrieval speeds. Hamilton et al. state that if the cache sizes are not too big and there are only a few objects then an optimal replacement policy can be determined by solving a dynamic programming problem.

Psounis and Prabhakar [452] mention that the LRU cache replacement algorithm has been known to perform badly when it is employed in Web caches. This has prompted the aggregated application of measures such as the recency and frequency of use, the size, and the cost of retrieving a document, in order to develop strategies for cache replacement. Despite the fact that these strategies result in considerable betterment in the hit rate and also cause latency diminution, nevertheless, they often require sophisticated data structures. One way to dispense with such complex data structures is to employ a randomized algorithm for approximating them. Psounis and Prabhakar develop a randomized scheme and state that it performs well when contrasted with extant schemes and is also advantageous in some respects.

Starobinski and Tse [511] study probabilistic methods for Web caching. They develop randomized policies for managing Web caches. Their policies require constant time to deal with a hit or a miss. Their analysis involves probabilities and demonstrates that the performance of their algorithms is almost on par with an optimal off-line algorithm.

Vakali [530] discusses a history-based approach for designing cache replacement algorithms, wherein, a record of the history of cached objects is utilized for aiding cache replacement decisions. Vakali describes four cache replacement algorithms HLRU, HSLRU, HMFU and HLFU. These four cache replacement algorithms are history-based variants of the LRU, Segmented LRU, Most Frequently Used (expels most frequently requested objects from the cache) and the LFU cache replacement algorithms. The HLRU, HSLRU, HLFU algorithms are reported to exhibit better hit ratios and byte hit ratios when compared to their counterparts. In another article [531], Vakali discusses the application of the ideas of evolutionary computing to design cache replacement algorithms. Also refer [529].

Williamson and Busari [553] develop a synthetic Web proxy workload generator known as ProWGen for studying the sensitivity of proxy cache replacement policies to chosen Web workload characteristics. By using synthetic workloads generated by ProWGen, they demonstrate the relative sensitivity of three widely used cache replacement algorithms: LRU, LFU-Aging (a variant of LRU), and Greedy-Dual-Size to Zipf slope (see Glossary for a definition of Zipf slope and also refer the chapter on Zipf's law), temporal locality, and correlation between popularity and file sizes. Williamson and Busari also demonstrate the comparative insensitivity of these algorithms to one-timers (documents that are not accessed more than once or potentially few times) and the heavy-tail index (see [7] for more information on heavy-tails). Williamson and Busari also study the variations in the performance of the three cache replacement algorithms.

Cohen et al. [139] state that caches are being populated in advance in order to increase the accessibility of popular Web objects. Web objects are pushed by push-servers to the proxy cache servers by means of a multicast-based distribution or other approaches. Cohen et al. mention that it is not easy to develop scheduling algorithms for push-servers. Such algorithms must find out which Web objects must be broadcast when. Cohen et al. formulate a cache pre-filling push problem and solve it by developing effective algorithms.

Irani [260] studies page replacement algorithms when the pages are of varying sizes. The algorithms have applicability for Web page replacement in Web caching. Irani studies two models for the cost of an algorithm on the basis of a request sequence. The goal of the first model is to minimize the number of page faults, whereas, in the second model the goal is to keep the total number of bits that have been read into the cache to a minimum. Both off-line and randomized algorithms are discussed for the two models.

Chen et al. [120] describe a cluster-based parallel Web server system known as p-Jigsaw (Jigsaw is a Web server developed by the World Wide Web consortium). It relies on cooperative caching to minimize disk accesses. Chen et al. also assess three cluster-aware cache replacement algorithms.

Haverkort et al. [237] describe a class-based LRU cache replacement algorithm that takes into account both size and recentness of objects. Their goal is to get a good mix of both big and small objects in the cache. Consequently, this in turn must help achieve good functioning for both small and big objects. Haverkort et al. claim that the implementation overheads of their algorithm are comparable to the traditional LRU algorithm.

5. Exercises

1 What are cache replacement algorithms? Explain their significance.

2 Mention popular cache replacement algorithms.

3 Investigate different ways of classifying such algorithms.

4 Compare and contrast LRU and LFU.

5 What are key based cache replacement algorithms?

6 What are the benefits of employing size as a criteria when performing cache replacement?

7 What are cost based cache replacement algorithms?

8 Compare and contrast traditional, key based and cost based cache replacement algorithms.

9 Investigate criteria other than size and cost for designing cache replacement algorithms.